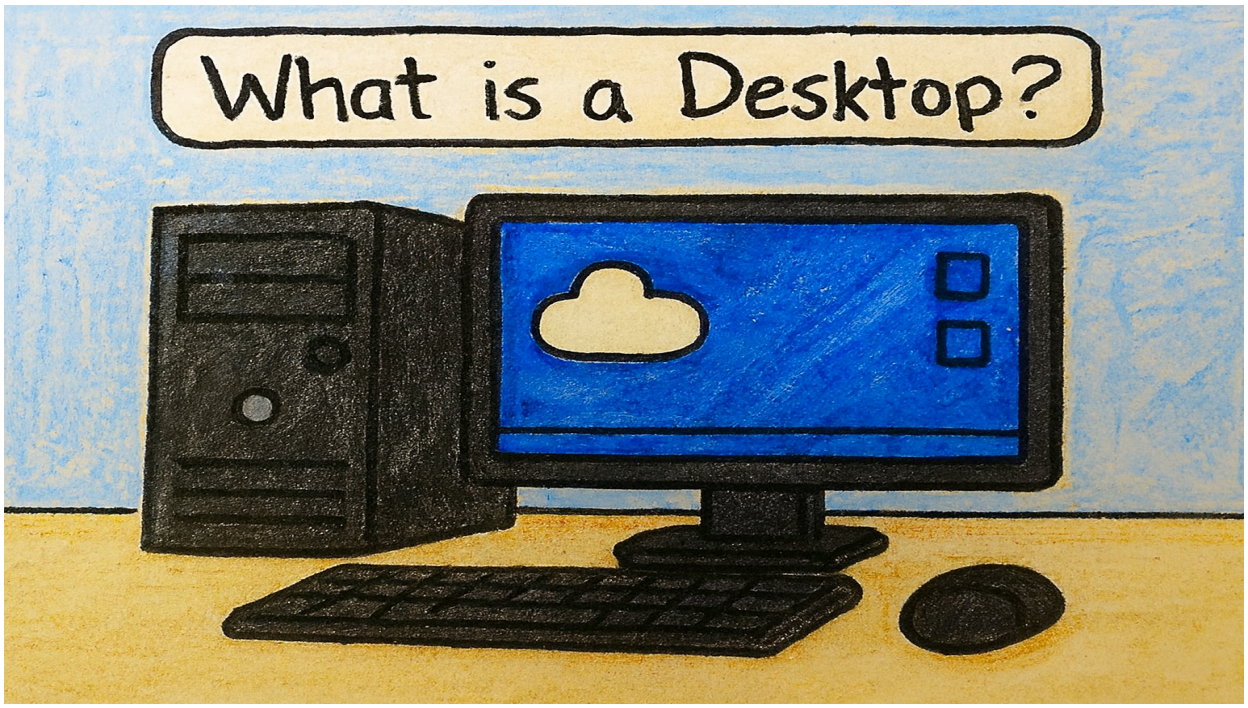

Class: 12th

Subject: Computer

Chapter 12: LOOP CONSTRUCTS

📍 Important MCQs:

1. Iteration in a program refers to:

(a) Selection of statements

(b) Repetition of statements ✅

(c) Termination of program

(d) Compilation process

2. Which one is the third type of program control structure in C?

(a) Sequence

(b) Selection

(c) Iteration ✅

(d) Jump

3. How many loop control statements are available in C?

(a) One

(b) Two

(c) Three ✅

(d) Four

4. Which of the following is NOT a loop statement in C?

(a) while

(b) do-while

(c) for

(d) if-else ✅

5. The while loop executes its body:

(a) At least once

(b) Only once

(c) As long as the condition is true ✅

(d) Until the program ends

6. In a while loop, the condition is checked:

(a) After executing the loop body

(b) Before executing the loop body ✅

(c) In the middle of loop execution

(d) At the end of program

7. A variable whose value controls the number of iterations is called:

(a) Counter variable

(b) Constant variable

(c) Loop control variable ✅

(d) Local variable

8. In a while loop, the loop control variable is usually:

(a) Declared inside the loop only

(b) Initialized inside the loop

(c) Initialized outside the loop body ✅

(d) Never initialized

9. What will happen if the condition in a while loop is false initially?

(a) Loop will execute once

(b) Loop will execute infinitely

(c) Loop body will not execute at all ✅

(d) Program will crash

10. The body of a while loop is enclosed within:

(a) Parentheses ( )

(b) Square brackets [ ]

(c) Semicolon ;

(d) Curly braces { } ✅

11. In a do-while loop, the test condition is checked: (a) Before executing the loop body

(b) During compilation

(c) At the end of the loop body ✅

(d) Inside the header only

12. The main advantage of a do-while loop is that it:

(a) Executes faster

(b) Uses less memory

(c) Executes the loop body at least once ✅

(d) Never checks condition

13. Which loop always executes its body at least one time?

(a) while

(b) for

(c) do-while ✅

(d) if

14. The do-while loop is mostly used when:

(a) Number of iterations is fixed

(b) Data must be validated after input ✅

(c) Condition is known in advance

(d) No condition is required

15. Which punctuation mark is compulsory at the end of do-while loop?

(a) Comma (,)

(b) Colon (:)

(c) Semicolon (;) ✅

(d) Full stop (.)

16. If the semicolon is omitted at the end of do-while loop, the result will be:

(a) Logical error

(b) Runtime error

(c) Syntax error ✅

(d) Infinite loop

17. The condition (state != 'w' && state != 'd') becomes TRUE when:

(a) state is 'w'

(b) state is 'd'

(c) state is neither 'w' nor 'd' ✅

(d) state is empty

18. A compound condition using logical AND (&&) is TRUE only if:

(a) Any one sub-condition is true

(b) All sub-conditions are true ✅

(c) All sub-conditions are false

(d) First condition is false

19. In a while loop, the loop body:

(a) Always executes once

(b) Executes after condition is false

(c) May or may not execute ✅

(d) Executes before condition check

20. In do-while loop, program control first:

(a) Checks the condition

(b) Executes loop body ✅

(c) Terminates the loop

(d) Skips the loop

21. Which loop is preferred by most programmers due to flexibility?

(a) while

(b) do-while

(c) for ✅

(d) goto

22. How many expressions are there in a for loop statement?

(a) One

(b) Two

(c) Three ✅

(d) Four

23. Which expression of for loop is executed only once?

(a) Test condition

(b) Increment expression

(c) Initialization expression ✅

(d) Loop body

24. Which expressions in a for loop are optional?

(a) Test condition only

(b) Initialization and increment/decrement expressions ✅

(c) All expressions

(d) None

25. Which expression in a for loop cannot be omitted?

(a) Initialization expression

(b) Increment expression

(c) Loop condition ✅

(d) Loop body

26. A nested loop is defined as:

(a) An infinite loop

(b) A loop without a condition

(c) A loop inside another loop ✅

(d) Multiple conditions in a loop

27. As the level of nesting increases, the program complexity:

(a) Decreases

(b) Remains the same

(c) Increases ✅

(d) Becomes zero

28. Which loops can be nested in C language?

(a) Only for inside for

(b) Only while inside while

(c) Only do-while inside do-while

(d) Any type of loop inside any other loop ✅

29. In nested loops, the outer loop is the loop that:

(a) Executes last

(b) Is placed inside another loop

(c) Contains the inner loop ✅

(d) Has no condition

30. In nested loops, the variables of the inner loop are:

(a) Initialized only once

(b) Never initialized

(c) Re-initialized for each iteration of the outer loop ✅

(d) Declared outside the loop

31. A sentinel loop terminates when:

(a) The counter becomes zero

(b) The condition becomes true

(c) A special sentinel value is entered ✅

(d) Memory becomes full

32. In the marks input program, a negative number is used as a sentinel because:

(a) Negative marks are allowed

(b) It reduces memory usage

(c) Students cannot have negative marks ✅

(d) Zero causes syntax error

33. Why is zero not a suitable sentinel value in the marks program?

(a) Zero causes runtime error

(b) Zero is not an integer

(c) Zero can be a valid mark ✅

(d) Zero stops program execution

34. Division by zero results in:

(a) Syntax error

(b) Compile-time error

(c) Runtime error ✅

(d) Logical error

35. Type casting is used to:

(a) Change the variable permanently

(b) Temporarily convert one data type into another ✅

(c) Increase execution speed

(d) Remove fractional values

36. The expression (float) total_students is used to:

(a) Declare a float variable

(b) Convert an integer to float temporarily ✅

(c) Store decimal values permanently

(d) Avoid loop execution

37. The goto statement performs:

(a) A conditional jump

(b) A function call

(c) An unconditional transfer of control ✅

(d) Loop termination

38. A label used with a goto statement must be:

(a) In another function

(b) In a header file

(c) In the same function ✅

(d) Outside the program

39. In the square root program, when a negative number is entered, the control:

(a) Terminates the program

(b) Displays an error message

(c) Transfers to the label positive ✅

(d) Skips execution

40. The major disadvantage of using the goto statement is that it:

(a) Uses more memory

(b) Slows down execution

(c) Makes the program difficult to understand and maintain ✅

(d) Causes syntax errors

📍 Important Short Questions:

1. What is iteration in C programming?

Answer:

👉 Iteration is the repetition of a statement or a group of statements in a program until a specific condition is satisfied.

Example: Printing numbers from 1 to 10 using a loop.

2. What is a loop in C?

Answer:

👉 A loop is a control structure that allows a set of statements to be executed repeatedly based on a condition.

Example: Using a while loop to display student roll numbers.

3. What is a while loop?

Answer:

👉 A while loop is a loop control statement that repeatedly executes its body as long as the given condition remains true.

Example: Printing numbers until a counter becomes greater than 10.

4. What is a loop control variable?

Answer:

👉 A loop control variable is a variable whose value controls the number of times a loop executes.

Example: The variable count in while(count <= 10).

5. Where is the loop control variable initialized and updated in a while loop?

Answer:

👉 In a while loop, the loop control variable is initialized before the loop starts and is updated inside the loop body.

Example:

count = 1; (initialization)

count++; (update inside loop)

6. What is a do-while loop?

Answer:

👉 A do-while loop is a loop control structure in C in which the loop body is executed first and the condition is checked at the end, so the loop executes at least once.

Example: Reading user input at least one time.

7. Why is a do-while loop executed at least once?

Answer:

👉 Because the condition is checked after executing the loop body, not before it.

Example: Displaying a menu before checking user choice.

8. Where is the condition checked in a do-while loop?

Answer:

👉 The condition is checked at the end of the loop body.

Example: do { } while(condition);

9. What is the main syntactical difference between while and do-while loops?

Answer:

👉 The do-while loop ends with a semicolon, while the while loop does not.

Example: Missing semicolon in do-while causes a syntax error.

10. Why is the semicolon mandatory in a do-while loop?

Answer:

👉 Because the do-while loop statement ends after the condition, and omitting the semicolon causes a syntax error.

Example: } while(x < 10);

11. What is a compound condition?

Answer:

👉 A compound condition is a condition formed by combining two or more conditions using logical operators.

Example: (state != 'w' && state != 'd')

12. When does a compound condition using AND (&&) become true?

Answer:

👉 A compound condition using AND becomes true only when all sub-conditions are true.

Example: Both (state != 'w') and (state != 'd') must be true.

13. When is a do-while loop preferred over a while loop?

Answer:

👉 A do-while loop is preferred when the loop body must be executed at least once.

Example: Validating user input.

14. What is a for loop in C?

Answer:

👉 A for loop is a loop control statement that uses initialization, condition, and increment/decrement expressions in a single line.

Example: Printing numbers from 1 to 10.

15. Which expressions are optional in a for loop?

Answer:

👉 The initialization expression and increment/decrement expression are optional, but the condition is mandatory.

Example: for(; count <= 10;)

16. What is a nested loop?

Answer:

👉 A nested loop is a loop inside the body of another loop. It allows repeated execution of the inner loop for each iteration of the outer loop.

Example: A while loop inside a for loop.

17. Can any type of loop be nested in C?

Answer:

👉 **Yes**, any loop (while, do-while, for) can be nested inside another loop.

Example: for loop containing a do-while loop.

18. What happens to inner loop variables in nested loops?

Answer:

👉 Inner loop variables are re-initialized every time the outer loop starts a new iteration.

Example: Counter of inner loop resets when outer loop increments.

19. What is a sentinel loop?

Answer:

👉 A sentinel loop is a loop that continues until a special sentinel value is entered.

Example: Reading student marks until a negative number is entered.

20. Why should zero not be used as a sentinel value for marks?

Answer:

👉 Because zero can be a valid mark, so using it could terminate the loop incorrectly.

Example: Negative numbers are used instead.

21. Why is type casting used in average calculation?

Answer:

👉 Type casting converts an integer to float temporarily to get an accurate fractional result.

Example: average = sum / (float) total_students;

22. What is the syntax of the goto statement?

Answer:

👉 The goto statement unconditionally transfers control to a labeled statement in the same function.

Example:

goto label;

label: statement;

23. Can a label in goto be used in another function?

Answer:

👉 No, the label must be in the same function where the goto statement is used.

Example: positive: label in the square root program.

24. How does the goto statement work in the square root program?

Answer:

👉 If a negative number is entered, control jumps to the label positive to ask the user for a valid positive number.

Example: goto positive;

25. What is the drawback of using goto statements?

Answer:

👉 Using goto excessively can make the program confusing and hard to maintain, creating “spaghetti code.”

Example: Multiple jumps in a large program reduce readability.

📍 Important long questions:

🌟 Q.1: Explain the while loop with syntax, working, and example

❖ **Answer:**

The while loop in C is used to repeatedly execute a set of statements as long as a given condition is true. It is useful when the number of iterations is not known in advance.

Syntax:

```
while (condition) {
    statements;
}
```

condition ⟶ A logical expression evaluated before each iteration.

statements ⟶ Body of the loop, executed only if the condition is true.

Working / Flow:

1. Initialize the loop control variable outside the loop.

2. Check the condition before executing the loop body.

   - If true, enter the loop.
   - If false, exit the loop.

3. Execute the loop body (statements inside { }).

4. Update the loop control variable inside the loop (increment or decrement).

5. Repeat steps 2–4 until the condition becomes false.

Example: Print numbers from 1 to 10

```
#include <stdio.h>

int main() {

   int count = 1;          // Step 1: Initialize loop control variable

   while (count <= 10) {   // Step 2: Check condition

      printf("%d\n", count);  // Step 3: Execute body
```

```
        count++;                // Step 4: Increment
    }
    return 0;
}
```

Explanation of Example:

1. count is initialized to 1.
2. The condition (count <= 10) is checked before entering the loop.
3. The loop body prints the value of count.
4. count is incremented by 1 after each iteration.
5. Loop repeats until count becomes 11, at which point the condition is false and the loop exits.

Key Points for Exam:

- The loop control variable is initialized outside the loop.
- Condition is checked before executing the body.
- The body may not execute at all if the condition is false initially.
- Always update the loop control variable to avoid infinite loops.

🌟 Q.2: Explain the do-while loop with syntax, example, and difference from while loop

❖ Definition:

The do-while loop in C is a post-tested loop, meaning the loop body is executed first and then the condition is checked. Because of this, the loop always executes at least once, even if the condition is false initially.

Syntax:

```
do {
    statements;
} while (condition);
```

- statements ⟶ Code inside the loop body, executed first.
- condition ⟶ Logical expression checked after executing the body.
- Ends with a semicolon (;), which is mandatory.

Working / Flow:

- Control enters the loop body directly.
- All statements inside the body are executed.
- After execution, the condition is evaluated.

- If the condition is true, control returns to the beginning of the loop body.
- If the condition is false, the loop terminates and control moves to the next statement after the loop.

Example: Force user to enter a valid telephone status (w for working, d for dead)

```
#include <stdio.h>

int main() {

    char state;

    do {

        printf("Enter telephone status (w for working, d for dead): ");

        scanf(" %c", &state);

    } while (state != 'w' && state != 'd');  // Repeat until valid input

    printf("Valid input entered: %c\n", state);

    return 0;

}
```

Explanation of Example:

1. Loop body executes first, asking the user to input a character.
2. Condition (state != 'w' && state != 'd') is evaluated after input.
3. If input is invalid, the loop repeats and asks again.
4. If input is valid, the loop exits and the program prints the entered value.

Difference from while loop:

1. **In a while loop**, the condition is checked first, so the body may not execute at all if the condition is false.
2. **In a do-while loop**, the body executes first, so it executes at least once.
3. The do-while loop requires a semicolon at the end, unlike the while loop.
4. Use do-while when you want to guarantee at least one execution, such as menu displays or input validation.

Key Points for Exam:

- The loop body always executes at least once.
- Ideal for input validation and menus.
- Do not forget the semicolon after a while(condition);.

- Condition is checked after the body, unlike the while loop where it is checked before.

🌟 Q.3: Explain the for loop with syntax, working, and example

❖ **Definition:**

The for loop in C is a pre-tested loop used to execute a set of statements repeatedly a known number of times. It is especially useful when the number of iterations is known in advance. The for loop is preferred by many programmers due to its compact structure.

Syntax:

```
for(initialization; condition; increment/decrement) {
   statements;
}
```

1. Initialization ⟶ Sets the starting value of the loop control variable. Executed only once at the start.
2. Condition ⟶ A logical expression tested before each iteration. The loop continues only if it is true.

3. Increment/Decrement ⟶ Updates the loop control variable after each iteration.
4. Statements ⟶ Body of the loop, executed if the condition is true.

Note: All expressions are optional except the condition, which is mandatory. If omitted, an infinite loop may occur.

Working / Flow:

1. The initialization expression is executed once at the beginning.
2. The condition is evaluated.
    - **If true**, the loop body executes.
    - **If false**, the loop terminates and control moves to the next statement after the loop.
3. **After executing the body,** the increment/decrement expression updates the loop control variable.
4. The condition is checked again, and the loop repeats until the condition becomes false.

Example: Print numbers from 1 to 10

```
#include <stdio.h>

int main() {
```

```
    int count;

    for(count = 1; count <= 10; count++) {  // Step 1: Initialize, Step 2: Condition, Step 4: Increment

        printf("%d\n", count);              // Step 3: Execute body

    }

    return 0;

}
```

Explanation of Example:

- count is initialized to 1.
- Condition (count <= 10) is checked before each iteration.
- The body prints the value of count.
- count is incremented by 1 after each iteration.
- The loop stops when count becomes 11, and control moves to the next statement after the loop.

Optional Expressions:

- **Initialization can be omitted:** You can initialize the variable outside the loop.
- **Increment/Decrement can be omitted:** You can update the variable inside the loop body.

- **Condition cannot be omitted:** Without a condition, the loop will become infinite unless explicitly controlled inside.

Example of optional expressions:

```
int count = 1;

for(; count <= 10;) {

    printf("%d\n", count);

    count++;

}
```

- Initialization is done outside.
- Increment is done inside the loop.
- Condition is mandatory.

Key Points for Exam:

- Best used when the number of iterations is known.
- Compact structure combines initialization, condition check, and update in one line.
- Loop may execute zero or more times depending on the condition.

- Supports optional initialization and increment, making it flexible.

🌟 Q.4: Explain Nested Loops with Example and Flow

❖ Definition:

A nested loop is a loop inside the body of another loop. Nested loops are used when a task requires repeated iterations for each iteration of an outer loop.

- Any type of loop (for, while, or do-while) can be nested.
- The inner loop runs completely for every single iteration of the outer loop.
- Variables used in the inner loop are re-initialized each time the outer loop starts a new iteration.

Working / Flow:

1. The outer loop starts its first iteration.
2. The inner loop executes fully for the current outer loop value.
3. After the inner loop finishes, the outer loop variable is updated (incremented/decremented).
4. The inner loop executes again for the next outer loop iteration.

5. Steps 2–4 repeat until the outer loop condition becomes false.

Example: Print a triangle pattern using a while loop inside a for loop

```
#include <stdio.h>

int main() {

   int outer, inner;

   for(outer = 1; outer <= 5; outer++) {  // Outer loop controls rows

      inner = 1;                    // Inner loop variable initialized

      while(inner <= outer) {           // Inner loop prints stars

         printf("*");

         inner++;

      }

      printf("\n");                    // Move to next line after inner loop

   }
```

```
    return 0;
}
```

Explanation of Example:

- The outer loop runs from 1 to 5 (controls the number of rows).
- For each value of outer, the inner loop prints * symbols equal to the current value of outer.
- After the inner loop finishes, printf("\n") moves the cursor to the next line.
- The inner loop variable inner is re-initialized to 1 for each new iteration of the outer loop.

Output of this program:

```
*

**

***

****

*****
```

Key Points for Exam:

- Nested loops are used for patterns, tables, or multi-level iterations.
- Inner loop executes completely for each outer loop iteration.
- Be careful with initialization of the inner loop variable; otherwise, the loop may give wrong results or an infinite loop.
- Complexity increases with more levels of nesting, so it should be used wisely.

🌟 Q.5: Explain the Goto Statement with Syntax, Example, and Importance

❖ Definition:

The goto statement in C is used to perform an unconditional jump to a labeled statement within the same function. It allows the program to transfer control directly to a specific part of the code, skipping intermediate statements.

Syntax:

goto label;

label: statement;

- label is an identifier followed by a colon (:).
- The goto statement transfers control immediately to the labeled statement.
- Both the goto and label must be in the same function.

Working / Flow:

1. When the program encounters goto label;, it jumps directly to the labeled statement.
2. The statements in between are skipped.
3. The labeled statement is executed next, and normal execution continues after it.

Example: Program to calculate the square root of a positive number, handling negative input

```
#include <stdio.h>

#include <math.h>

int main() {

   float num;

positive:  // Label for goto
```

```
    printf("Please enter a positive number: ");
    scanf("%f", &num);
    if(num < 0) {
        goto positive;  // Jump back to label if number is negative
    } else {
        printf("Square root of %.2f is %.2f\n", num, sqrt(num));
    }
    return 0;
}
```

Explanation of Example:

- **The label positive:** marks the point to jump back to.
- If the user enters a negative number, the goto positive; statement sends control back to the label, asking the user again.
- If the number is positive, the program calculates and prints the square root.
- This ensures that the program handles invalid input repeatedly without using loops.

Importance of Goto:

- Simplifies code when a specific part needs to be re-executed under special conditions.
- Useful in error handling or breaking out of deeply nested loops.

Drawbacks:

- Overuse of goto can make code hard to read and maintain.
- Leads to “spaghetti code” if multiple jumps are scattered throughout the program.

Key Points for Exam:

- Always use goto sparingly.
- Only jumps within the same function.
- Provides a quick alternative to loops for simple re-execution scenarios.
- Loops are usually preferred for repetition, while goto is mainly for special cases.

🌟 Q.6: Compare While, Do-While, and For Loops

In C programming, loops are used to repeat a set of statements until a specific condition is met. There are three main types of loops: while, do-while, and for. Each loop has its unique behavior and purpose.

1. While Loop

Definition:

The while loop executes a block of statements as long as a specified condition is true.

Key Points:

- The condition is checked before entering the loop body.
- The loop may execute zero times if the condition is initially false.
- Useful when the number of iterations is unknown at the start.

Syntax:

```
while(condition) {
    statements;
}
```

Example: Print numbers from 1 to 10

```
#include <stdio.h>

int main() {
    int count = 1;              // Loop control variable initialized
    while(count <= 10) {        // Condition checked first
        printf("%d\n", count);  // Body executed if condition true
        count++;                // Increment inside the loop
    }
    return 0;
}
```

Flow:

1. Initialize count = 1
2. Check condition (count <= 10) ⟶ if true, enter body
3. Print count
4. Increment count
5. Repeat steps 2–4 until condition becomes false

2. Do-While Loop

Definition:

The do-while loop executes the loop body first and then checks the condition.

Key Points:

- The loop body is always executed at least once, even if the condition is false.
- Condition is checked after the body.
- Useful when input or action must occur at least once.

Syntax:

```
do {
    statements;
} while(condition);
```

Example: Force user to enter correct telephone state (w or d)

```
#include <stdio.h>
int main() {
    char state;
    do {
```

```
        printf("Enter state (w/d): ");
        scanf(" %c", &state);     // Read input
    } while(state != 'w' && state != 'd'); // Repeat if invalid
    return 0;
}
```

Flow:

1. Enter loop body ⟶ ask user input
2. Read input
3. Check condition (state != 'w' && state != 'd')
4. If true ⟶ repeat body
5. If false ⟶ exit loop

3. For Loop

Definition:

The for loop combines initialization, condition, and increment/decrement in one line. It is compact and preferred for loops with a known number of iterations.

Syntax:

for(initialization; condition; increment/decrement) {

```
    statements;
}
```

Example: Print numbers from 1 to 10

```
#include <stdio.h>
int main() {
    int count;
    for(count = 1; count <= 10; count++) {   // Initialization; condition; increment
        printf("%d\n", count);              // Body executed each iteration
    }
    return 0;
}
```

Flow:

1. Initialize count = 1 (executed only once)
2. Check condition (count <= 10)
3. Execute loop body ⟶ print count

4. Increment count
5. Repeat steps 2–4 until condition is false

✦ Comparison Summary

While Loop:

- Condition checked first
- May execute 0 or more times
- Good for unknown iterations

Do-While Loop:

- Body executed first, then condition checked
- Executes at least once
- Good when action must occur once

For Loop:

- Compact structure: initialization + condition + increment in one line
- Preferred for known iterations
- Easy to read and maintain

Exam Tip:

- Use while when the number of iterations is unknown.

- Use do-while when you must execute the body at least once.
- Use count-controlled loops like printing numbers from 1 to 100.

🌟 Q.7: Write a C program using nested loops to print a pattern and explain

❖ **Definition:**

A nested loop is a loop placed inside the body of another loop. The outer loop controls the number of rows, while the inner loop controls the number of columns or elements printed in each row.

Nested loops are useful for printing patterns like triangles, rectangles, or pyramids.

Example: Print a triangle pattern of numbers

1

12

123

1234

12345

C **Program**:

```
#include <stdio.h>

int main() {

   int i, j; // Loop control variables

   for(i = 1; i <= 5; i++) {       // Outer loop → controls rows

      for(j = 1; j <= i; j++) {   // Inner loop → controls columns

         printf("%d", j);        // Print numbers in each row

      }

      printf("\n");              // Move to next line after inner loop

   }

   return 0;

}
```

Explanation (Flow of Control):

1. Outer loop (i) starts from 1 to 5 ⟶ controls the number of rows.
2. Inner loop (j) starts from 1 to i → prints numbers in each row.

- On the first iteration of the outer loop, inner loop prints 1.
- On the second iteration, inner loop prints 12.
- This continues until the fifth row, printing 12345.

3. After the inner loop completes each row, printf("\n") moves the cursor to the next line.
4. Outer loop increments ⟶ inner loop re-initialized ⟶ process repeats.
5. When outer loop reaches 6 ⟶ condition false ⟶ loop terminates.

Key Points:

- Nested loops are used to print patterns or perform multi-level iterations.
- Inner loop variables are re-initialized each time the outer loop executes.
- **Control flow:** Outer loop ⟶ Inner loop ⟶ Print ⟶ Next line ⟶ Repeat.
- Can use any type of loop (for, while, do-while) for outer and inner loops.

Note:

This chapter is designed to provide a solid foundation of knowledge, with the goal of deepening understanding and encouraging further exploration of the subject. The content has been carefully selected to support effective learning and inspire students to engage with the topic more deeply.

Author: Muhammad Asghar

Purpose: To contribute to education by offering insightful, valuable content that enhances learning and understanding.

Copyright & Usage Policy